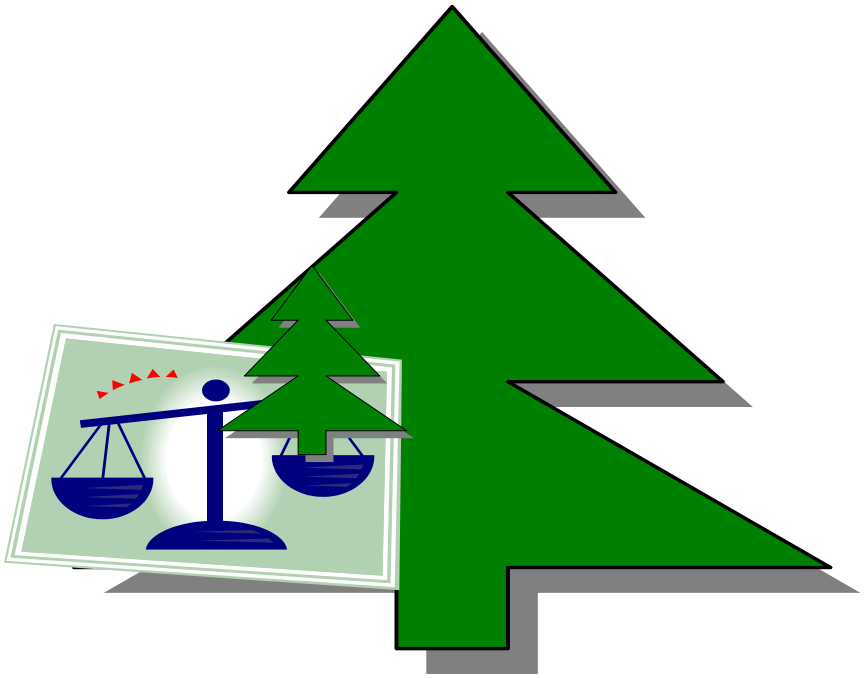
Meer over Binaire Zoek Bomen en varianten



Dr. D.P. Huijsmans
10 okt 2012
Universiteit Leiden, LIACS

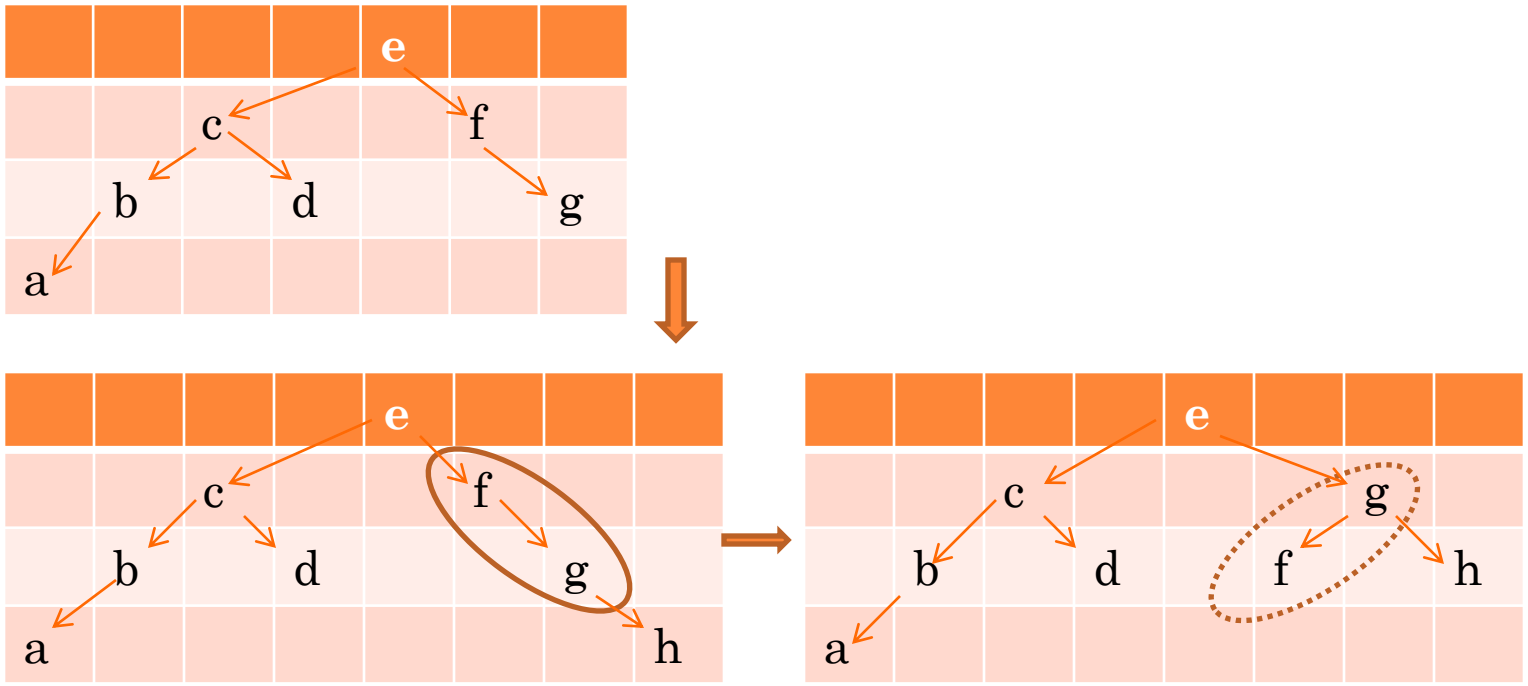
(Her)balanceren van BST

- Hoe goed is de balans binnen een boom?
- Hoe karakteriseren we een optimale balans?
- Hoe goed is de balans onder een knoop?
- Hoe kunnen we de balans lokaal verbeteren?
- Hoe wordt de balans verstoord t.g.v. insert/delete?
- Kunnen we deze aanpassingen niet automatisch in de ADT opnemen?

Verstoring balans door insert

- Toevoeging in rechter subtree:

Linksrotatie nodig



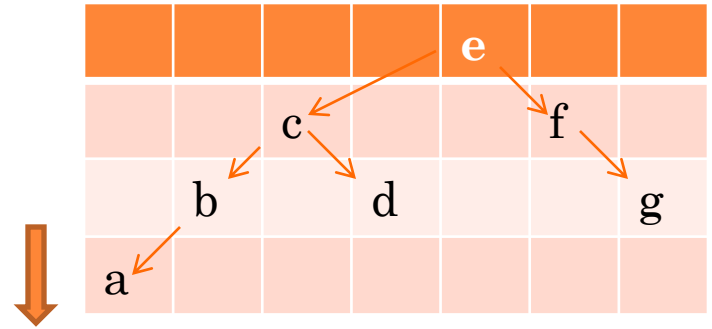
Balansfactoren:

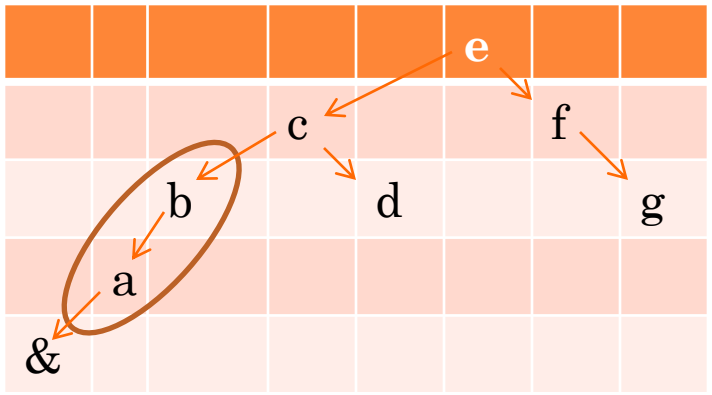
				0			
		-1			2		
	-1		0			1	
0							0

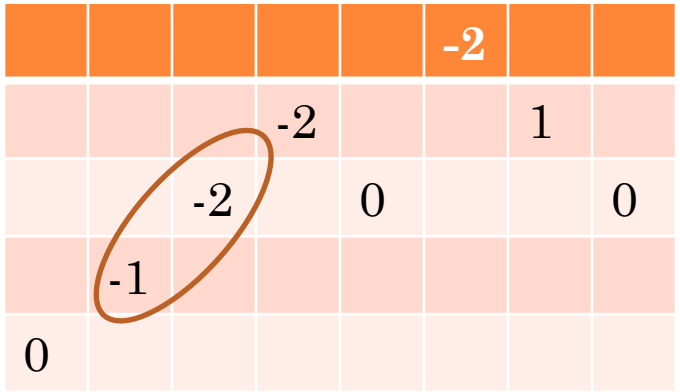
				-1			
		-1				0	
	-1		0		0		0
0							

Verstoring balans door insert

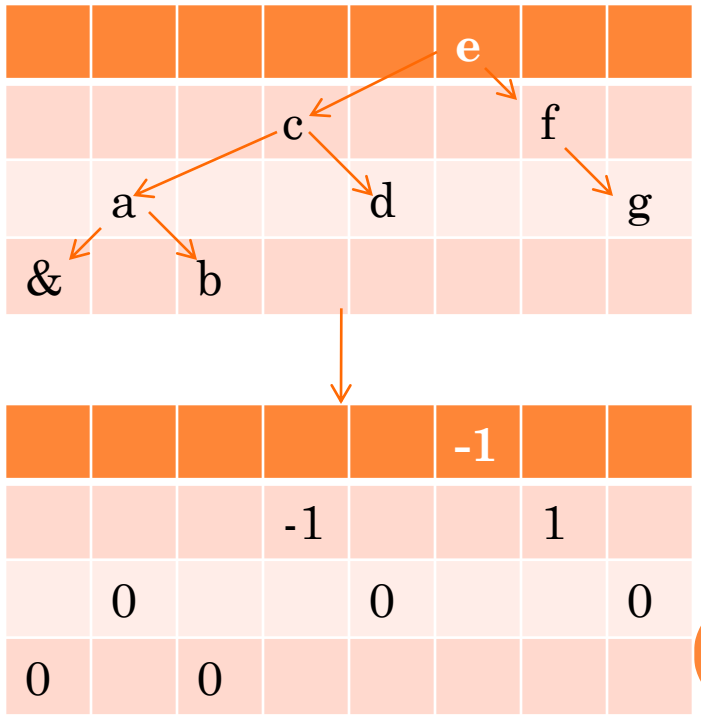
- Toevoeging in linker subtree (van b):





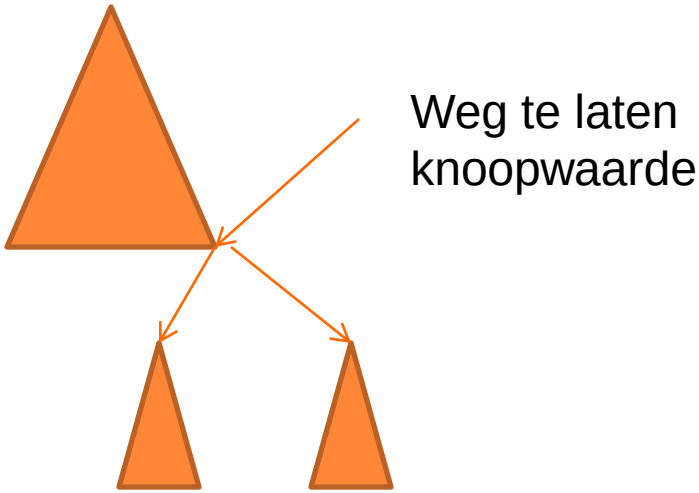


Na rechtsrotatie



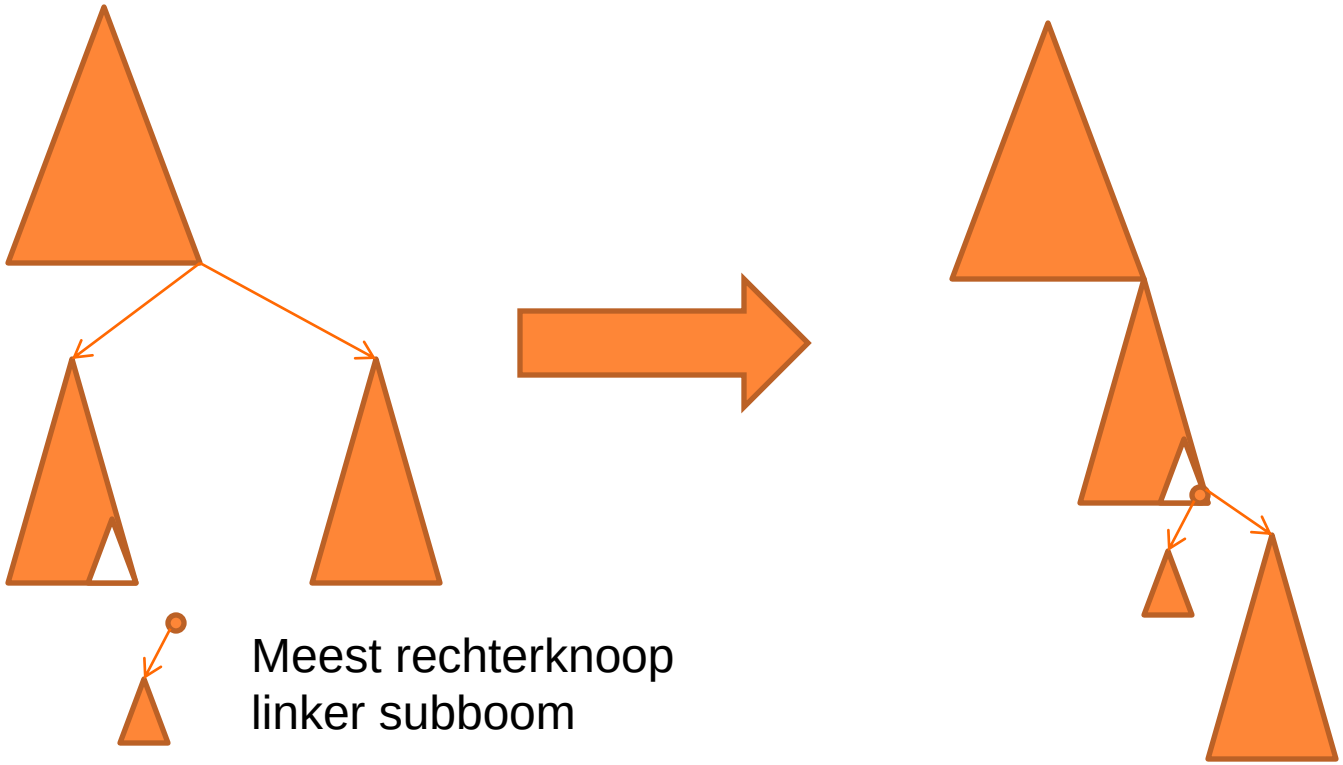
DELETE KEY UIT BINAIRE ZOEK BOOM

- Afgezien van een te deleten key die in een blad zit of maar 1 subboom heeft is delete alleen lastig
- Als de te deleten key 2 subbomen heeft:
- Hiervoor zijn twee alternatieve oplossingen bedacht:
- Delete by merging
- Delete by copying

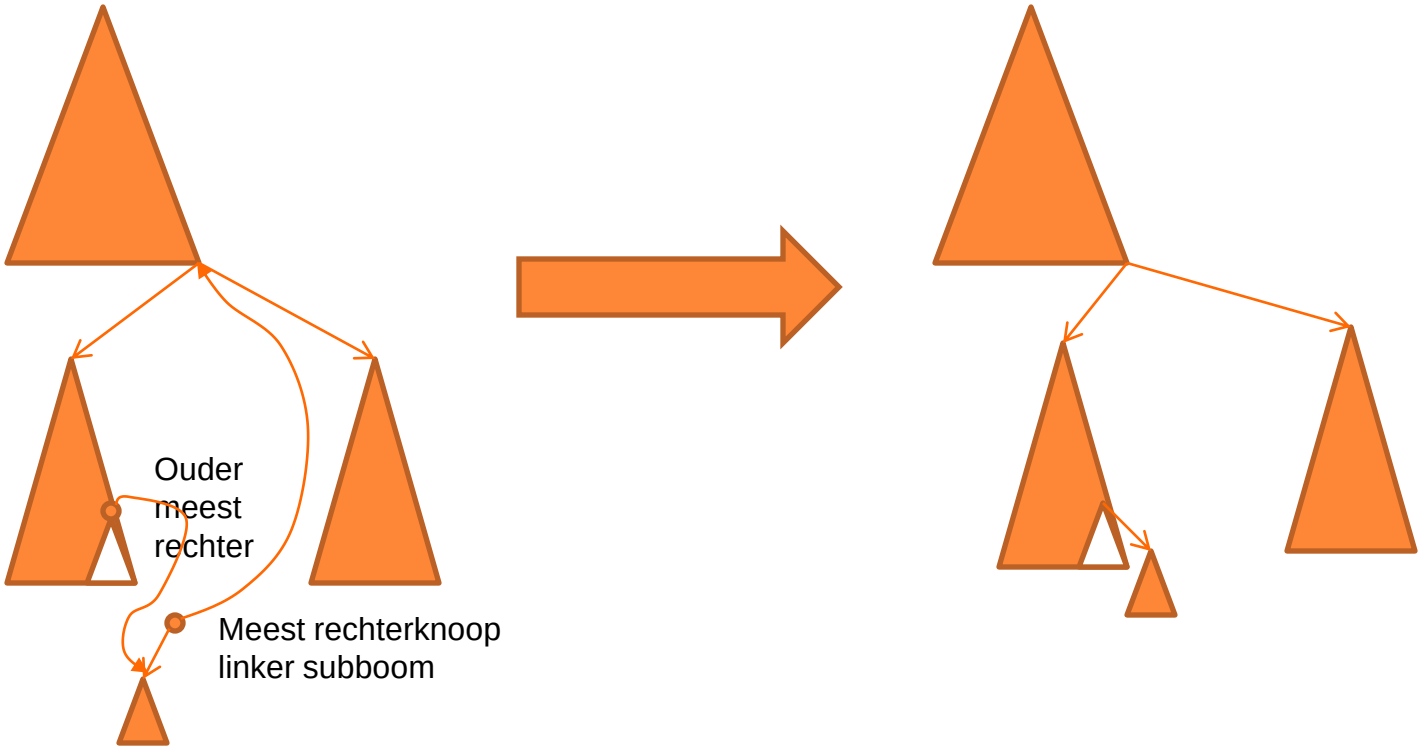


- In beide gevallen gaan
- We iets doen met de
- Knoop die qua sortering er vlak voor of vlak na zit

Delete by merging

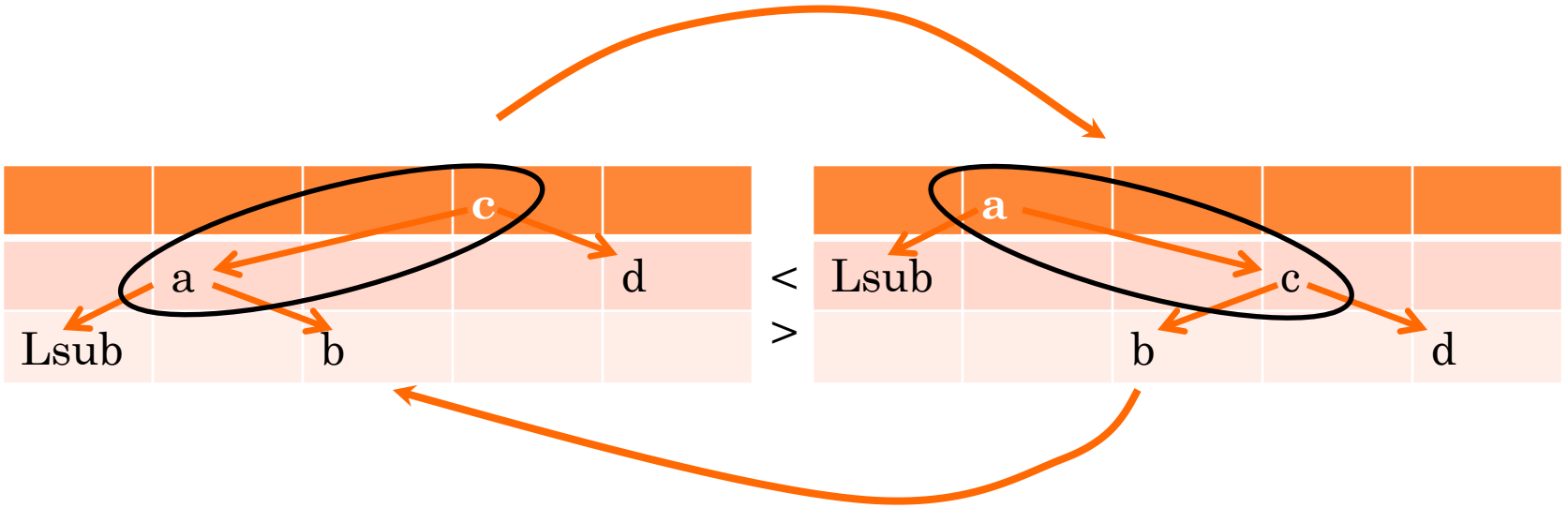


Delete by copying



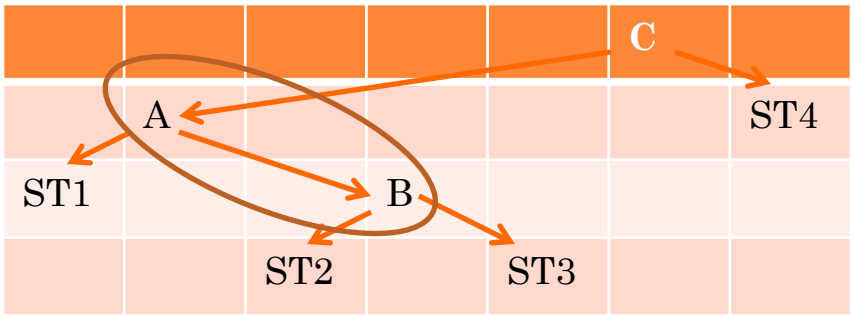
Knuth, Hibbard: Alterneren tussen meest rechter van linker subboom en meest linker van rechter subboom

Links- en rechtsrotatie zijn elkaars inverse/spiegelbeeld



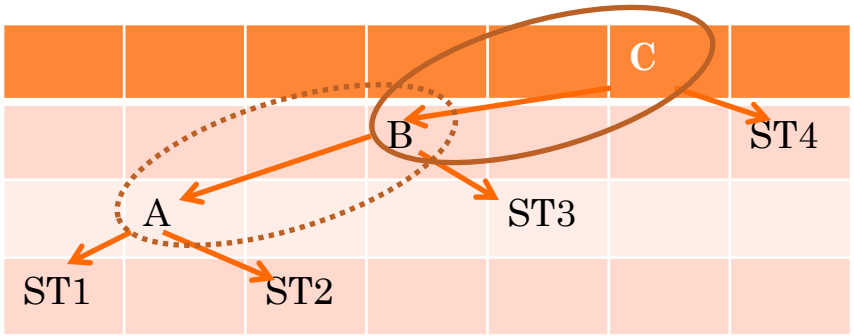
Dubbele (ZagZig) rotatie

start



ST1	A	ST2	B	ST3	C	ST4

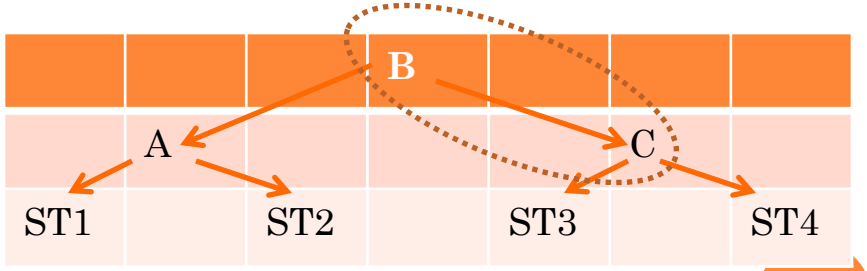
1xL



ST1	A	ST2	B	ST3	C	ST4

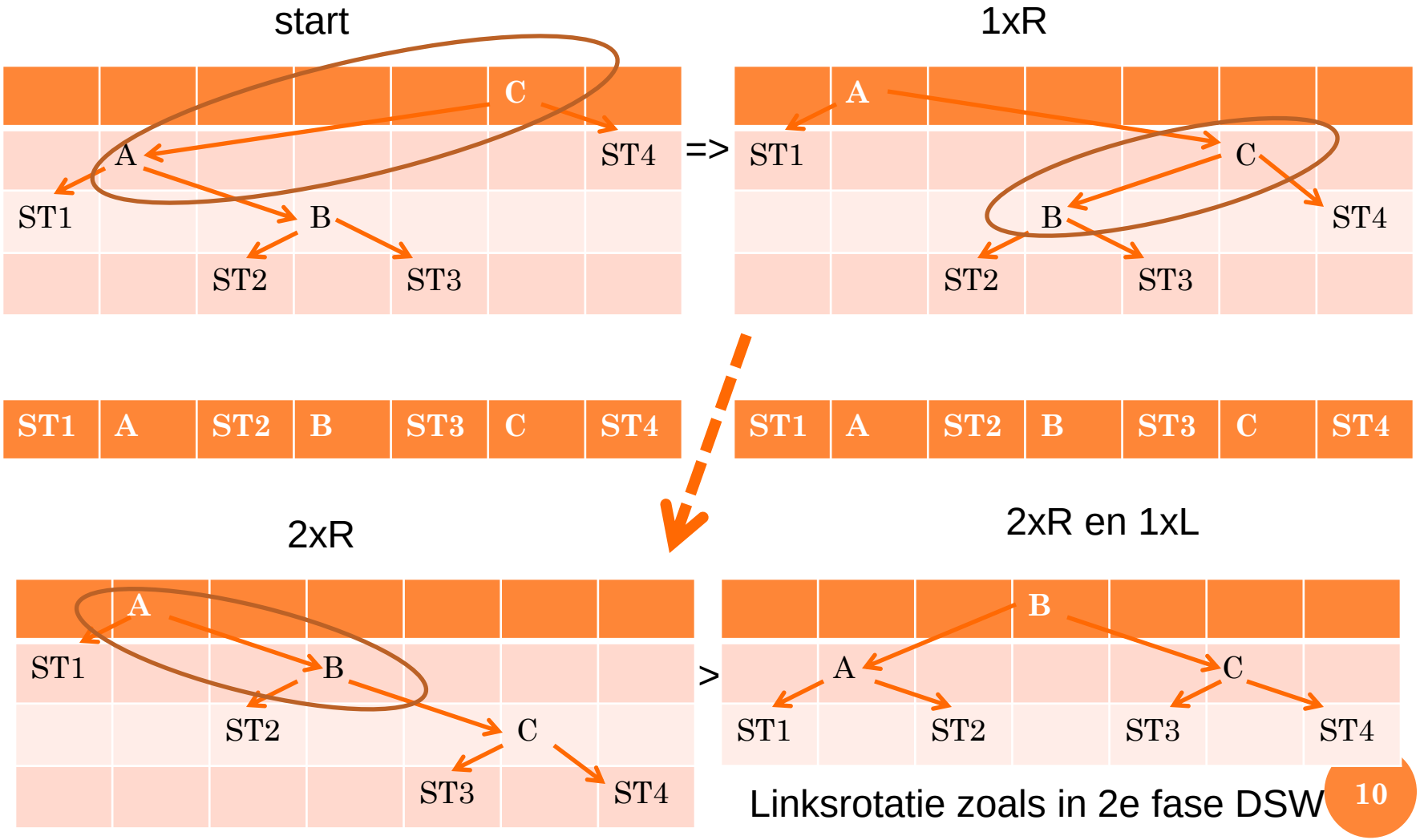
Dubbele (ZagZig) rotatie
Voldoet 1xL en 1xR

1xL en 1xR



Vergelijk met vorig college
2XR en 1xL waar voor zelfde resultaat 3 rotaties nodig

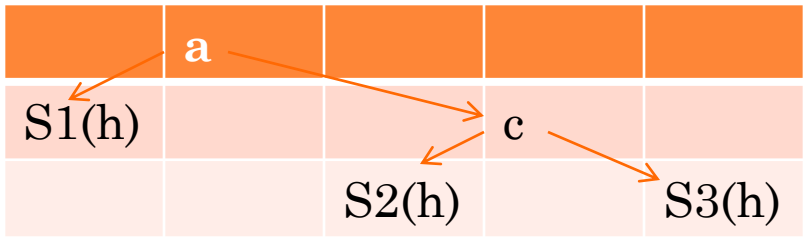
Oude DSW oplossing i.p.v. zagzig
Dubbele Rechts Rotatie en 1 x Links

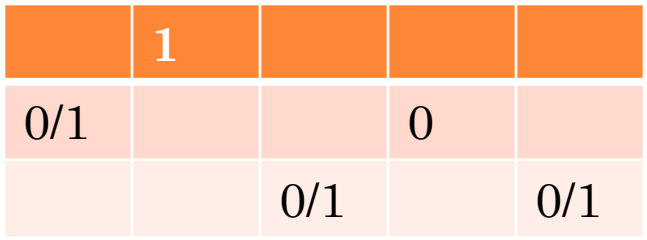


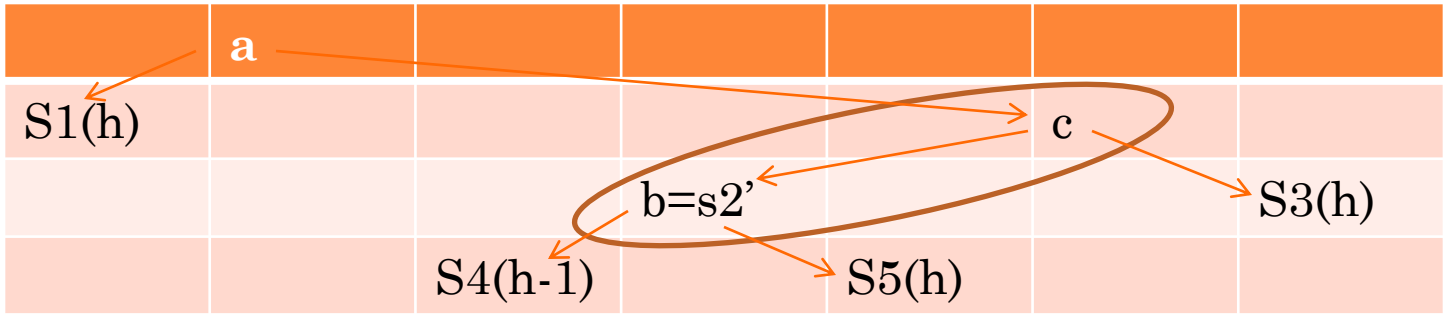
Linksrotatie zoals in 2e fase DSW

Locale backbone structuur zoals na 1e fase DSW

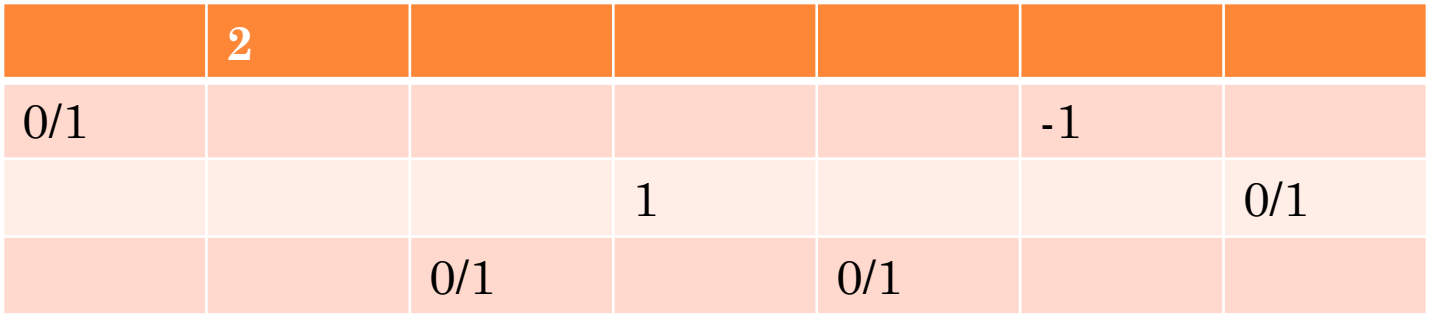
Ergste verstoring balans door insert



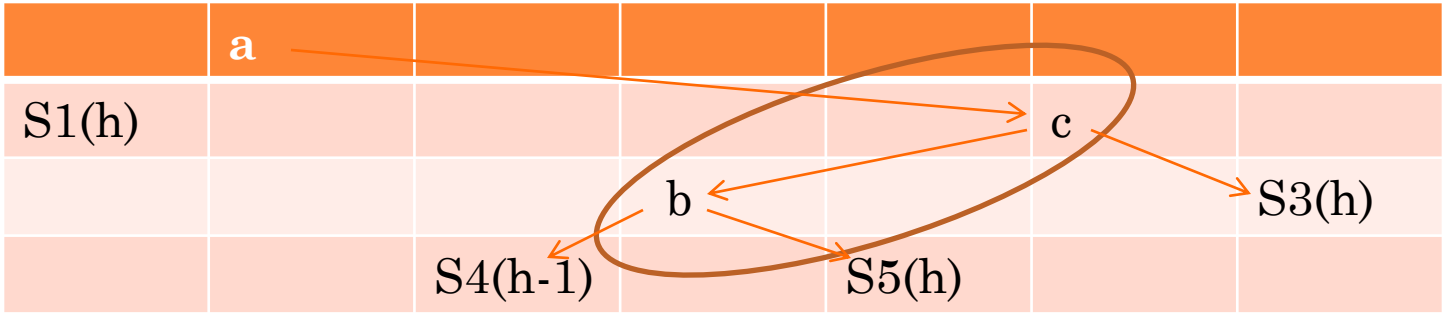




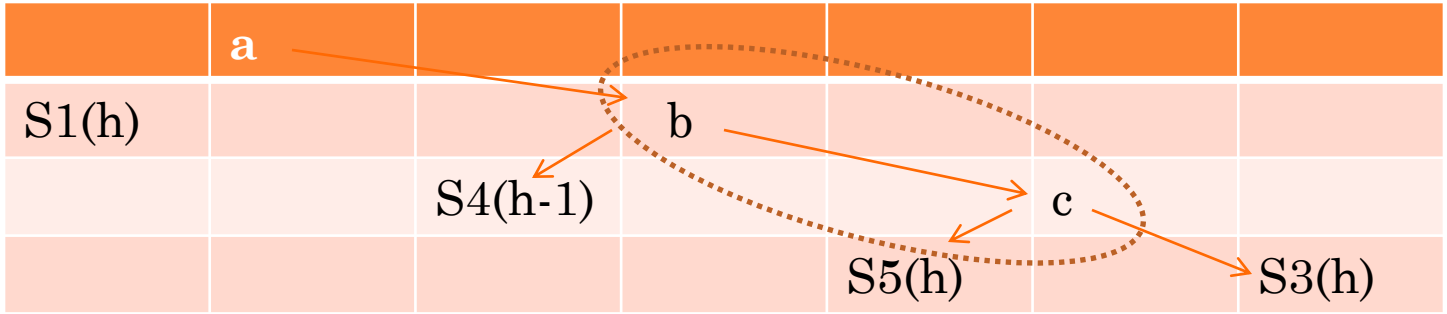
S2 heeft er een knoop bijgekregen (hier rechts in S5)



Dubbele rotatie nodig (eerst Zig)-1

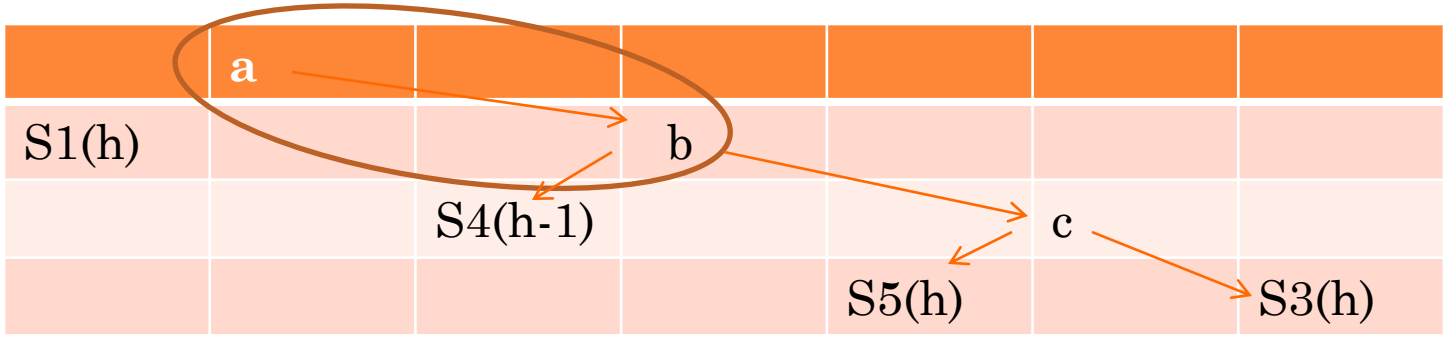


rechtsrotatie

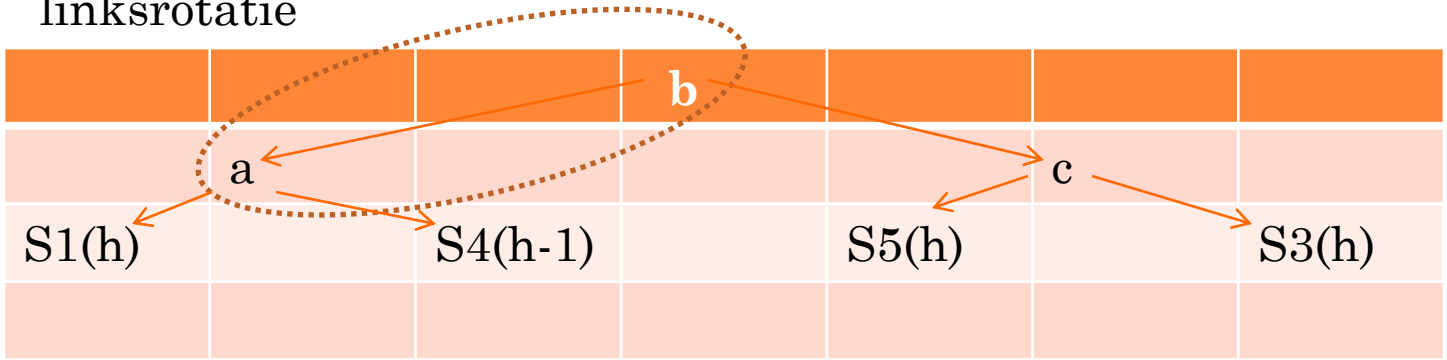


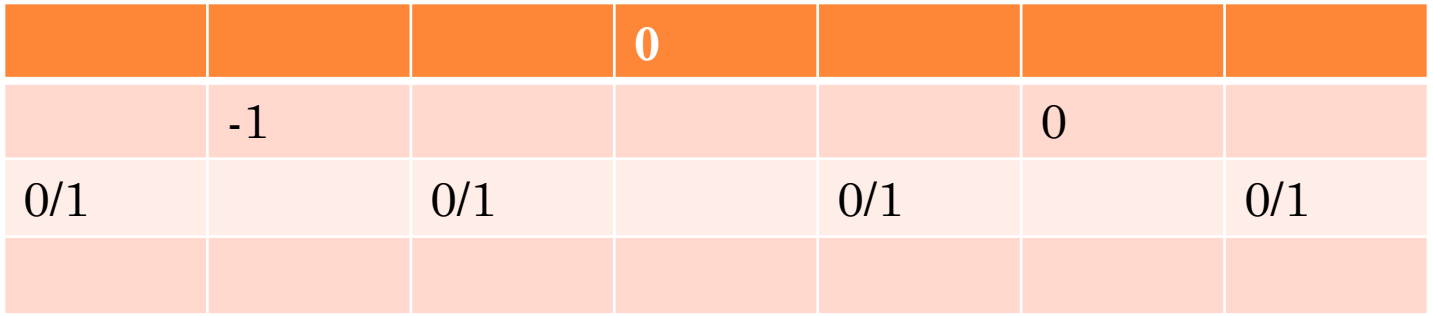
	2					
0/1			1			
		0/1			0	
				0/1		0/1

Gespiegelde zagzig=zigzag 1xR en 1xL Dubbele (dan zag) rotatie nodig-2



linksrotatie





AVL:Detectie of aanpassing nodig

- Na toevoeging knoop:
- Pad nieuwe knoop tot root knoop volgen en balansfactor gepasseerde knopen aanpassen
- Als alle balansfactoren langs pad [-1,0,1] klaar
- Zo niet, d.m.v. rotatie(s) balans verbeteren
  - Zig = rechtsrotatie
  - Zag= linksrotatie
  - Zigzag is ook een naam voor de vorm van de om te zetten structuur

AVL herbalancering na wijziging in subtree van een knoop

- In feite maakt het niet uit of je de situatie bekijkt t.o.v. een root knoop of lager in de boom, het aantal uitgangssituaties is maximaal beperkt tot:

subL	subR	bfknp	delR	insL	delL	insR
h-1	h	1	0	0	2	2
h	h-1	-1	-2	-2	0	0
h	h	0	-1	-1	1	1
h	h+1	1	0	0	2	2
h+1	h	-1	-2	-2	0	0

- Ook het R/L deleten is gelijk qua effect aan het L/R inserten; in 12 van de 20 gevallen geen wijziging nodig onder betrokken knoop; in 8 van de 20 gevallen wel

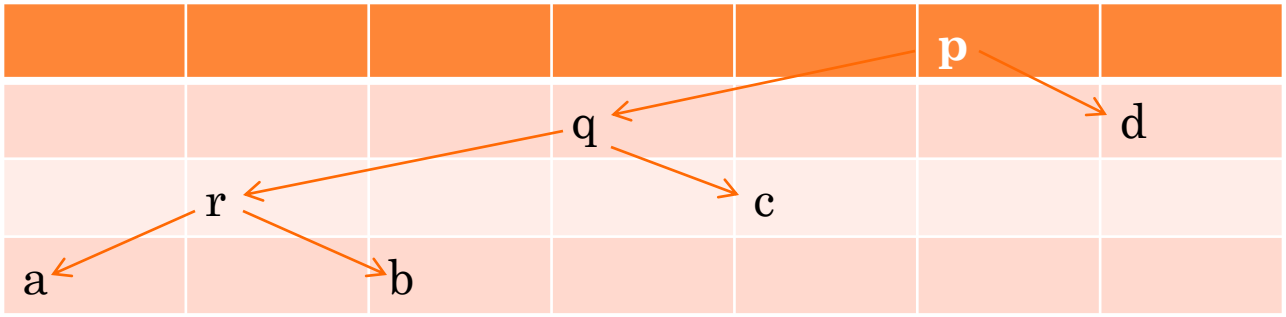
Samenvatting BST

- Blijft gesorteerd
- Effectief O(logN) zoeken mits goed gebalanceerd:
  - DSW:Globale balancering met optimaal resultaat
  - AVL: locale balancering met near-optimal result
- Vormen waarbij organisatie afhankelijk frequentie van toegang tot waarden in BST:
  - Self-restructuring trees
  - Splaying (zie boek Drozdek)

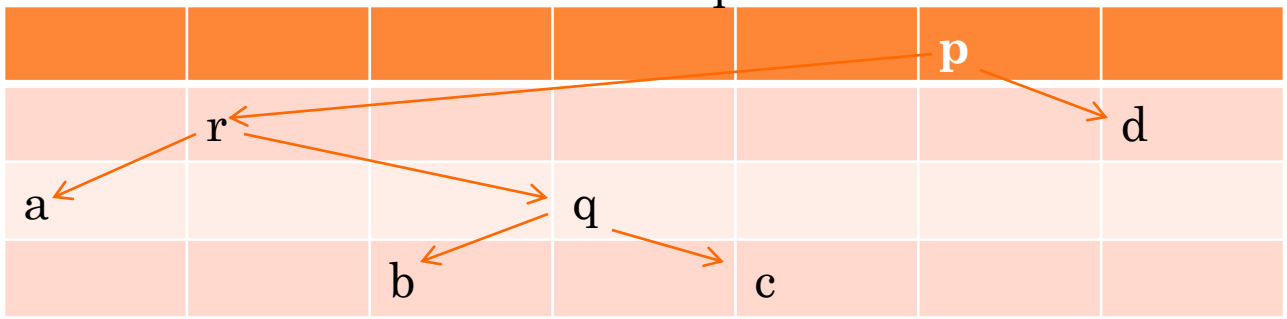
- Methode om te (her)balanceren:
  - links/rechtsrotatie

BST Variant
Self-Restructering Tree

- Doel is om (vaak?) bezochte elementen (tenzij al root) dichter bij de root te brengen via:
- Of: Een enkele rotatie van kind rond ouder als element in kind gebruikt
- Of: verplaats kind naar de root



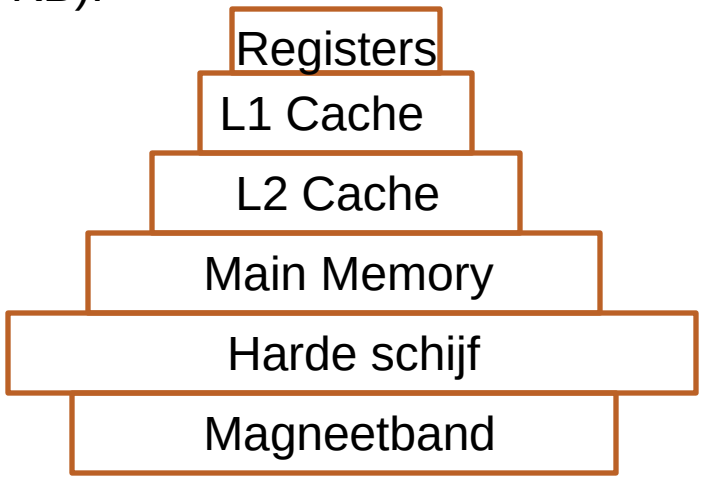
Voorbeeld:r bezocht roteer om q



Uitstapje naar File Systeem en Geheugenhierarchie

Extreem voorbeeld van afweging efficiency in ruimte en tijd

Typische grootte (in KB):		Toegangstijd in nsec:
1 (KB)	Registers	1 (nsec)
4.000	L1 Cache	2
40.000	L2 Cache	4
4.000.000	Main Memory	10
400.000.000	Harde schijf	1.000.000
4.000.000	Magneetband	10.000.000.000



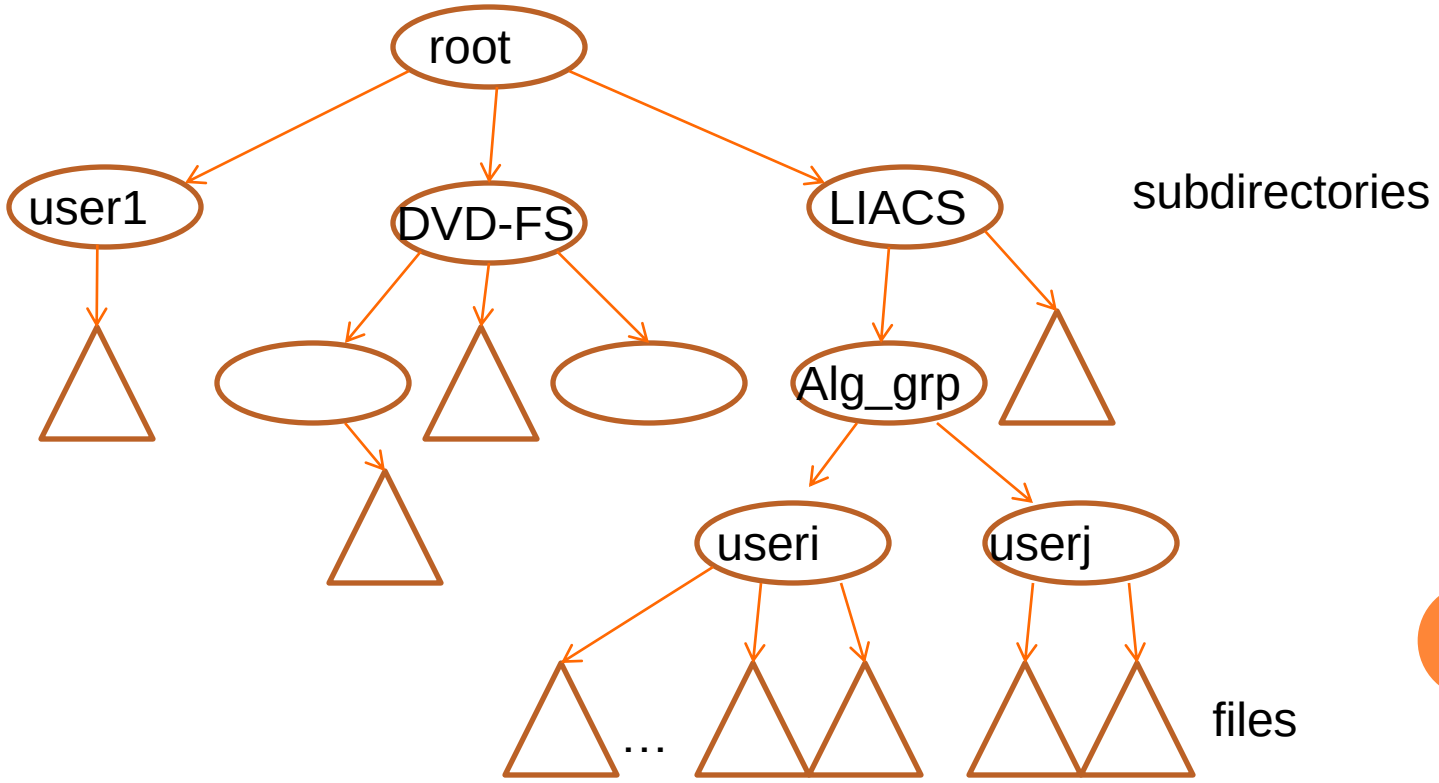
Goede zet: haal I/Ofile inhouden van een programma naar MM

Permanente opslag en toegang

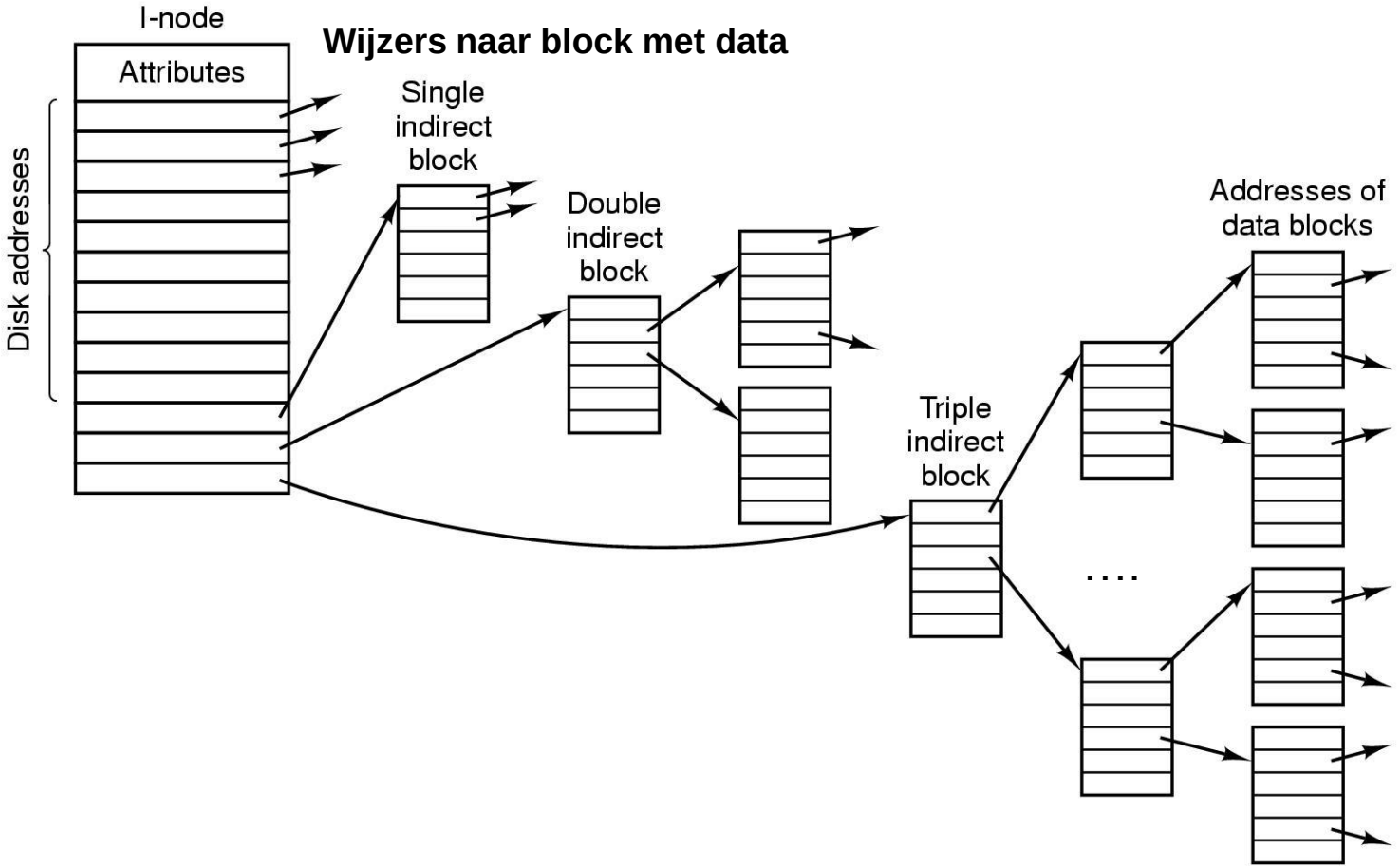
- Eenheid van opslag en hoe toegankelijk (RA of Serieel):
- Registers: 32/64 bit -> 4/8 bytes RA
- Caches: regels met 8-512 bytes RA
- MM: byte RA (1/2/4/8 bytes RA)
- HD: block met 512 bytes RA (byte in block serieel)
- MT: block met 512 bytes Serieel
- Doel1: hoe zorgen we er voor dat de data (bytes) die zo direct nodig zijn, al zo hoog mogelijk in deze geheugenhierarchie zitten?
- Doel2: hoe zorgen we ervoor dat gewijzigde data (bytes) uiteindelijk in het file systeem terechtkomen?

Data opslag en hierarchische naamgeving

- Data georganiseerd in files (bep. interne organisatie)
- Files (benoemd) vormen bladeren in
- (sub)directory structuur (boom structuur)



ADT_i-node
Unix i-node File organisatie



PAUZE

Variant BST
Heap

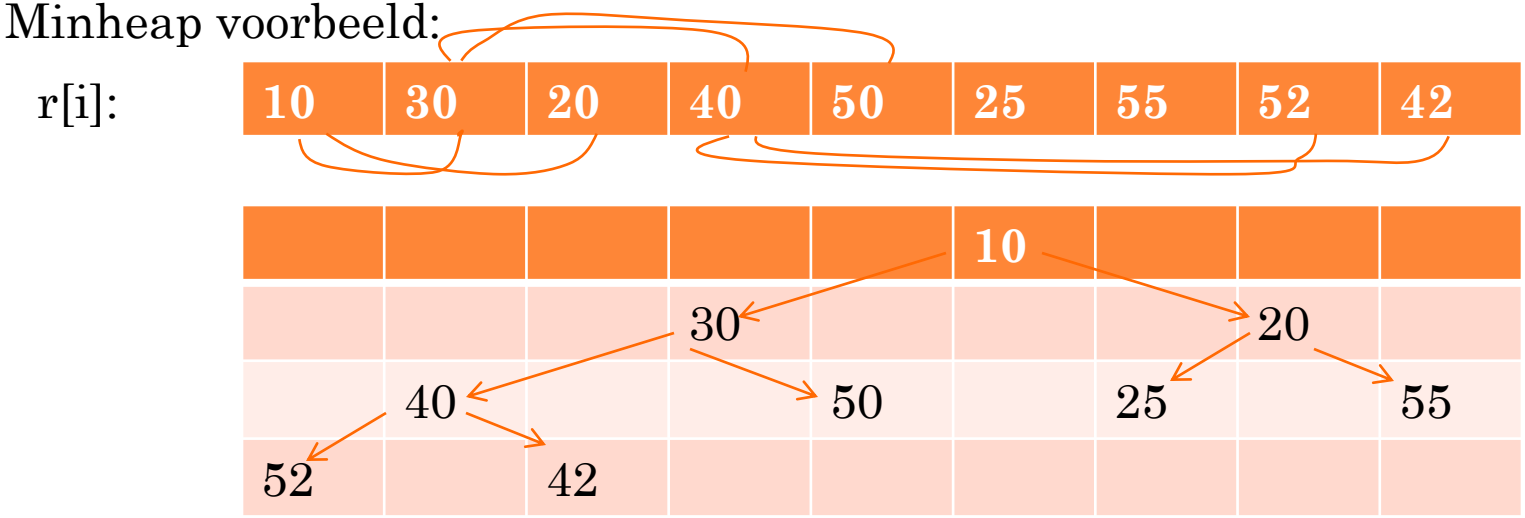
- Een Heap is een Binaire Boom met de volgende eigenschappen:
- Boom is perfect gebalanceerd
- Bladeren laatste niveau meest linkse positie
- (breadth first volledige vulling en van L -> R)
- MaxHeap:
- **De waarde in een knoop (niet blad) is >= die in kinderen**
- MinHeap:
- **De waarde in een knoop (niet blad) is <= die in kinderen**

- **Zie apart blad voor Heap specificatie en implementatie**

Array representatie Heap

- De Breadth first doorloop van de Binaire Heap en een array representatie van een heap zijn als volgt verbonden:
- Array r[i] met i [1..n] is een heap als
  - r[i] <r[2i] en r[i]<r[2i+1] (minheap) of <=
  - r[i] >r[2i] en r[i]>r[2i+1] (maxheap) of >=
  - Voordeel: ouder van r[j] is r[j div 2] direct beschikbaar

Minheap voorbeeld:

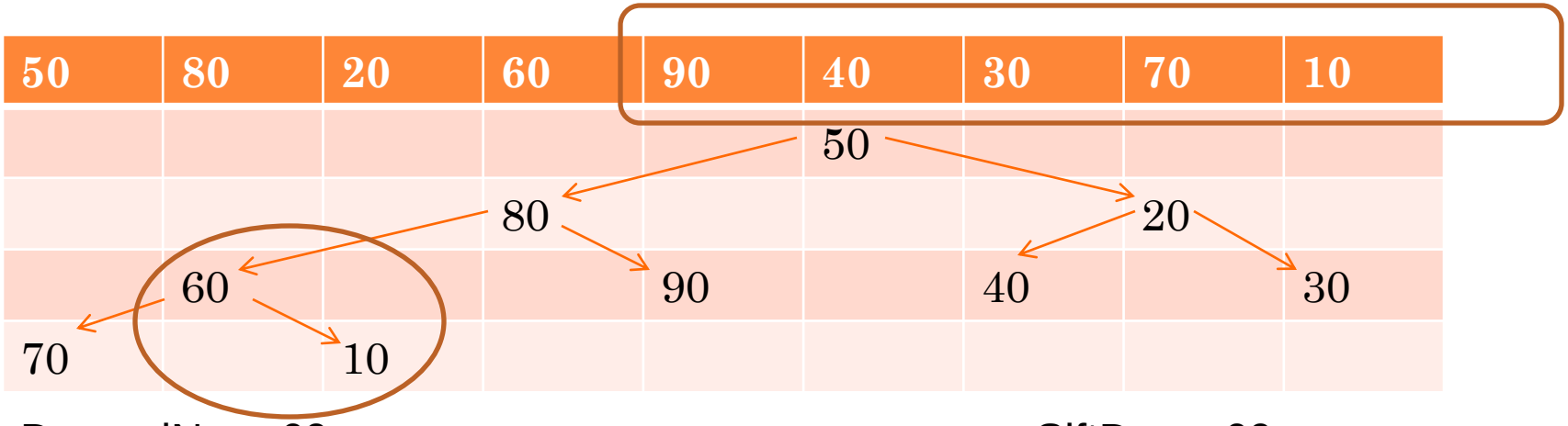


Van sequentie naar heap

- Een rij random gegenereerde integers in minheap vorm zetten gaat als volgt:
- Stap 1: vorm een complete binaire boom, breadth first van Links naar Rechts
- De bladeren in deze boom voldoen al aan de (min)heap conditie
- Stap 2: beginnend bij het laatste niet blad van de doorloop, wissel dat element om als tenminste 1 van de kinderen kleiner is; wissel om met de kleinste

MinHeap create voorbeeld-1

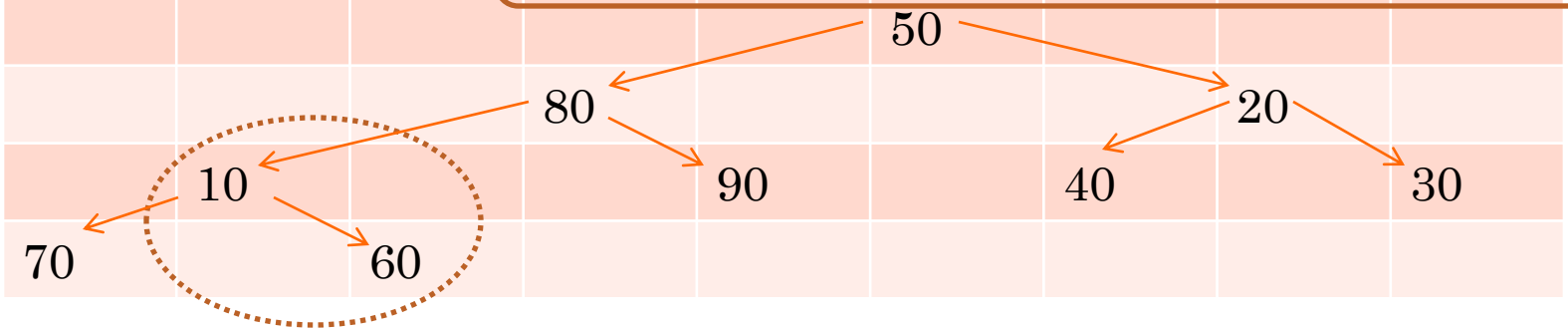
50	80	20	60	90	40	30	70	10



DwarrelNeer 60

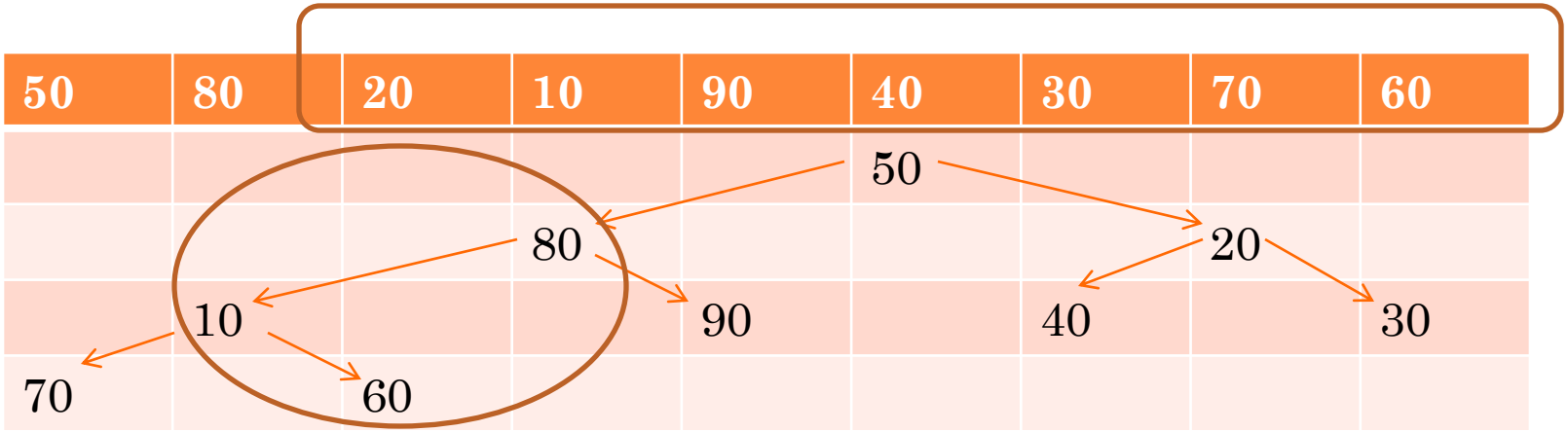
SiftDown 60

50	80	20	60	90	40	30	70	10



MinHeap create voorbeeld-2

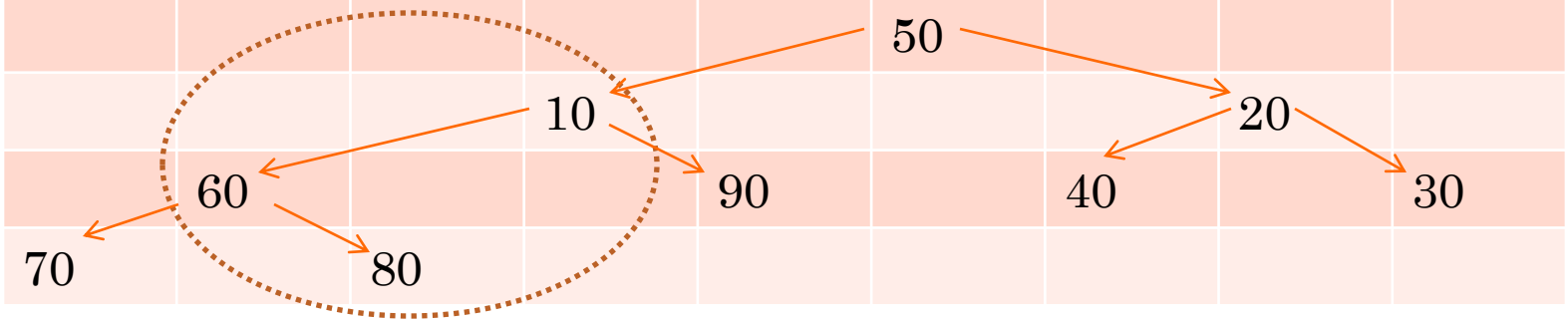
50	80	20	10	90	40	30	70	60



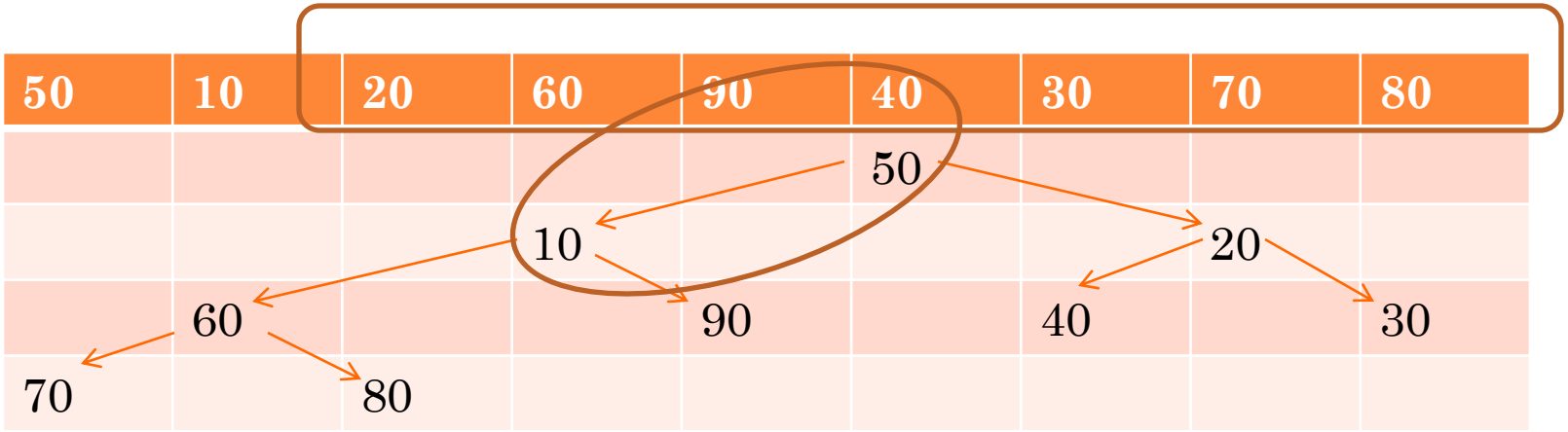
DwarrelNeer 80

SiftDown 80

50	10	20	60	90	40	30	70	80

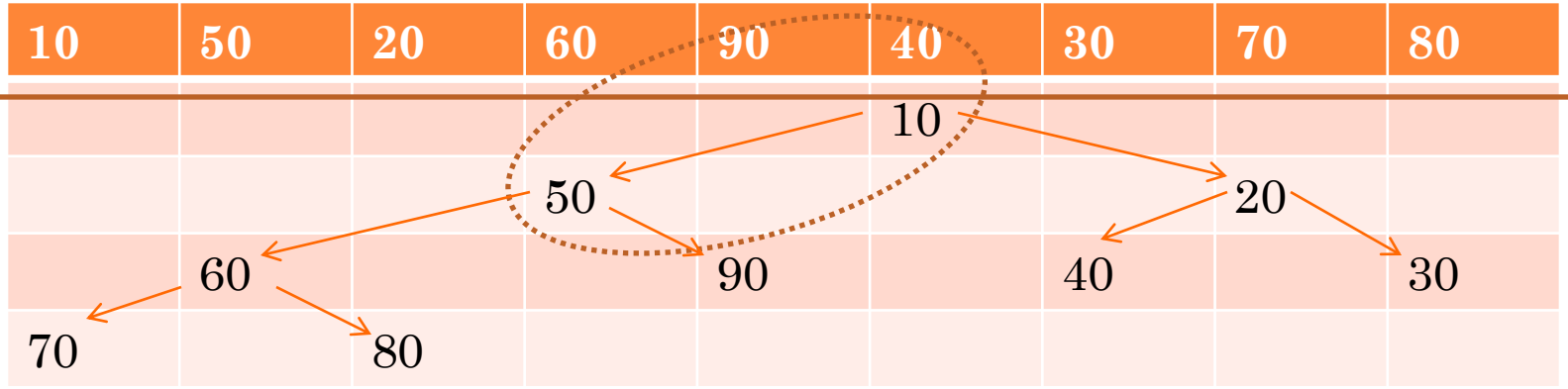


MinHeap create voorbeeld-3



DwarrelNeer 50

SiftDown 50



Nu gesorteerd als minheap: elk pad vanaf root -> blad is oplopend

Prioriteitsrij m.b.v. heap

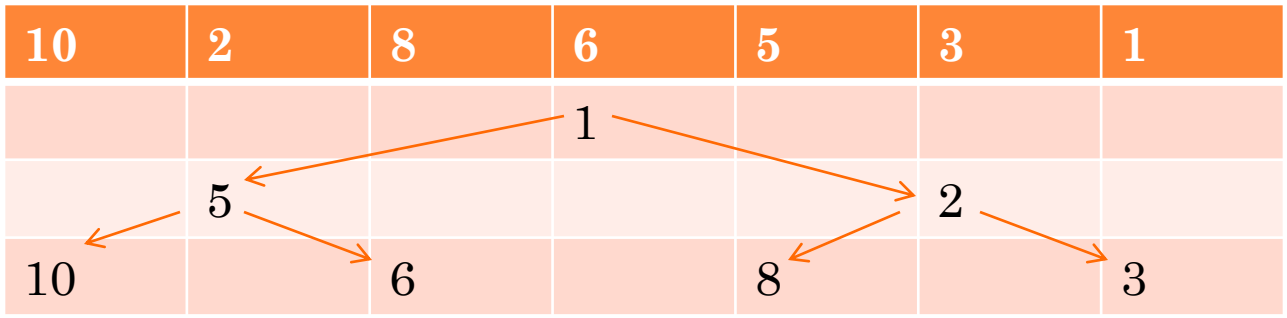
- Zie apart blad voor ADT_PrioriteitsRij specificatie en implementatie m.b.v. ADT_Heap
- Insert is achteraan, gevolgd door BorrelOp van laatste (SiftUp)
- Delete(bij Serve) is vooraan (root), gevolgd door laatste naar eerste, size-> size-1, en DwarrelNeer van eerste (SiftDown)

Heap of BST of gesorteerd array

- Op- of aflopend gesorteerd (min- of max heap)
- Heap is alleen gesorteerd langs elk simpel pad
- Gesorteerd array is tevens een Heap
- Root Heap en BST beide kleinste(grootste) element
- Minder sorteerwerk bij Heap bijwerken

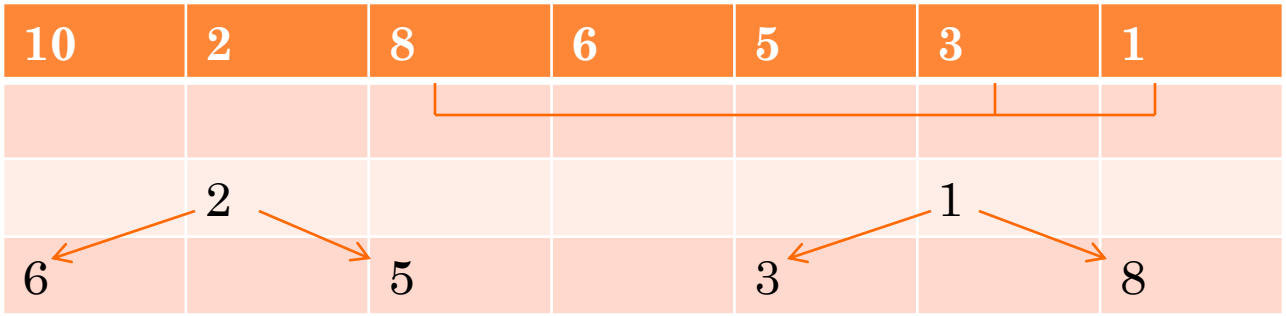
OPBOUWEN HEAP VANUIT ARRAY

- Twee manieren mogelijk:
- Methode 1: Top-down
- Vanuit array breadth first doorloop nemen
- Na elke knoop toevoeging :
- Check kind waarde langs simpel pad tot wortel met ouder voor heap voorwaarde
- Verwissel eventueel met ouder

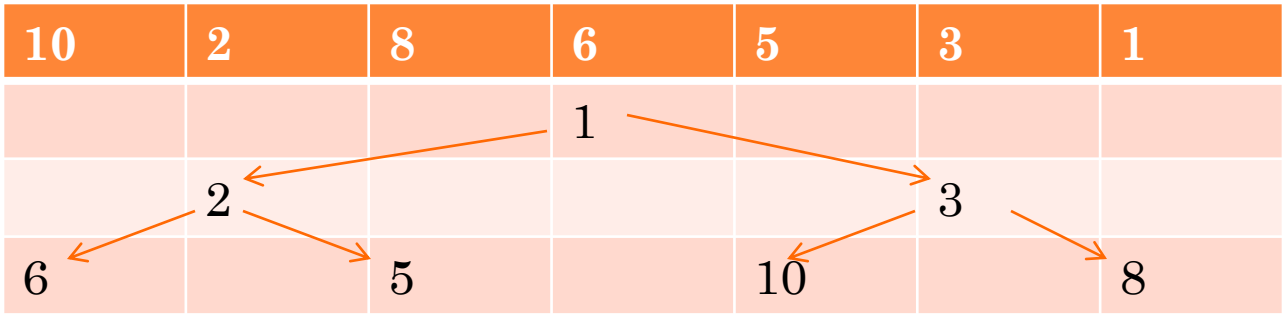


Methode 2: Bottum-up heap constructie

- Lees breadth first indexering maar nu van diepste niveau naar boven steeds twee kinderen plus ouder uit en zet langs simpel pad ouder,kinderen vorm in heap vorm m.b.v. verwisseling ouder kind (indien nodig)



Na 2 stappen



eindresultaat

ADT_FileSystem
File operaties

- Creëer een file binnen een File Systeem
- Delete een file uit het FS
- Open een file voor lezen
- Open een file voor schrijven
- Open een file voor lezen en schrijven
- Wijzig een file

- ADT_OS als verzameling ADT's: ADT_FS, ADT_JS…,ADT_NC

- En nog veel meer in college Operating Systemen

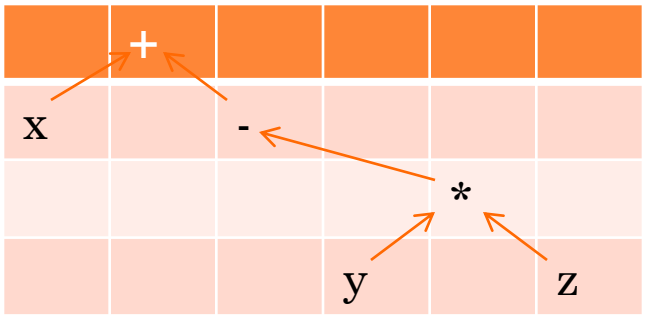
Expressie bomen
een laatste voorbeeld van binaire bomen

- Expressies bestaan uit waarden of variabelen waarop bewerkingen/operaties plaatsvinden (maximaal 2 invoeren/operandi tegelijk)
- Bij een ingewikkelde expressie is het zaak precies duidelijk te krijgen in welke volgorde bewerkingen plaatsvinden
- Bij bewerkingen is een voorkeursvolgorde afgesproken (mijnheer van Dalen….)
- M.b.v. haakjes kan een bepaalde afhandelingsvolgorde ook worden afgedwongen
- M.b.v. de juiste expressieboom kunnen deze haakjes worden weggelaten

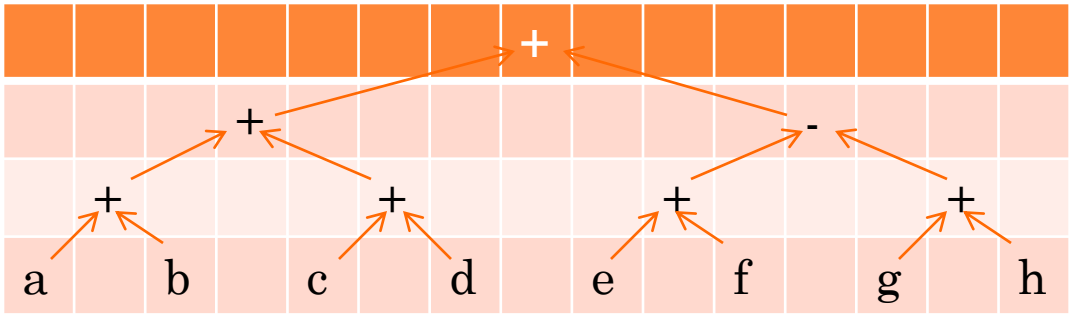
Maximaal 2 invoer waarden

- Machinecode instructies werken op 1 of 2 invoeren
- -> werking met binaire boom weer te geven:
- vb 1 p=x+(-(y*z))

- (min) is hier een unaire operator!

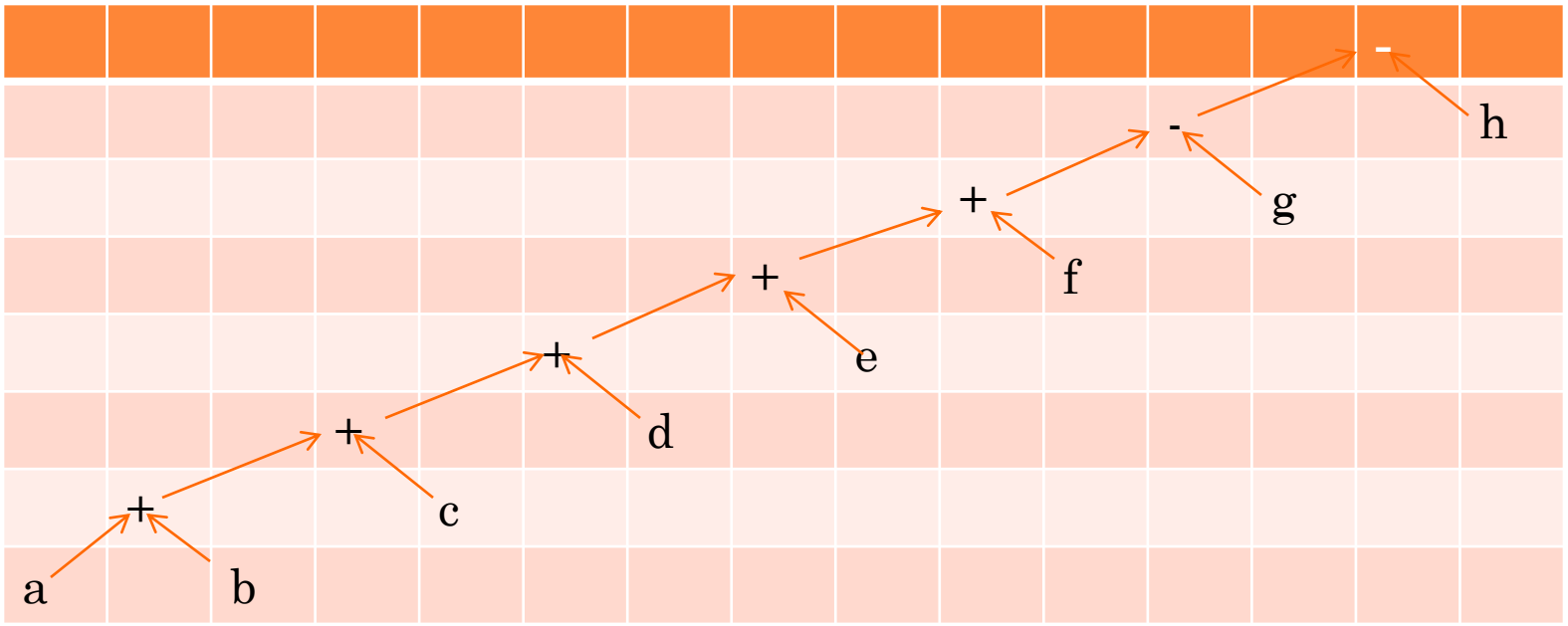


- vb 2 y=a+b+c+d+e+f-g-h (hoe in paren van max 2?)



Hoe expressies > 2 invoeren afhandelen?

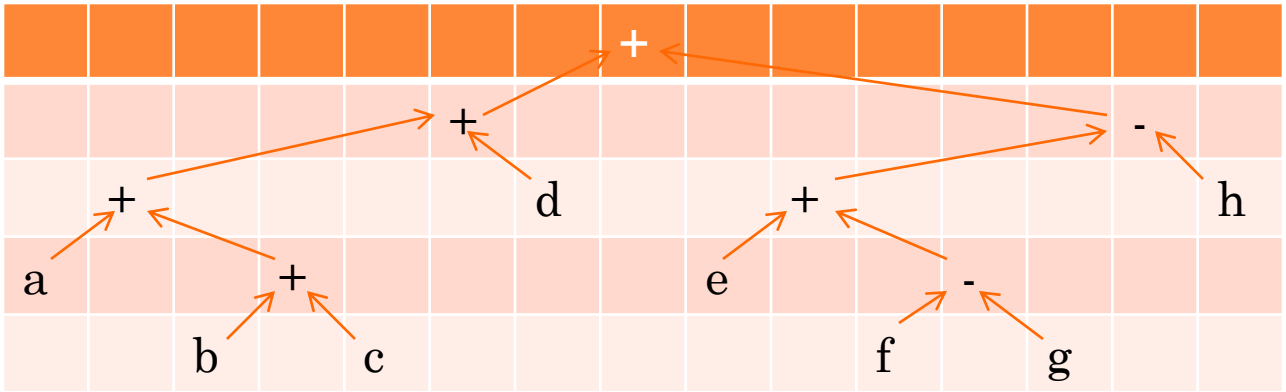
vb2 van L -> R



Omdat + en – op gelijke voet qua afhandeling bestaan er veel gelijkwaardige expressiebomen voor dit voorbeeld

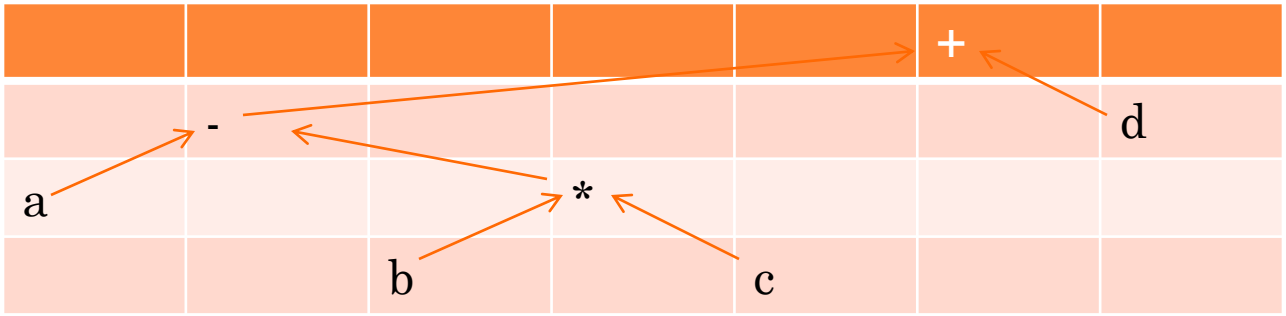
Afdwingen bewerkingsvolgorde m.b.v. haakjes

- Een veilige manier om een bepaalde afhandelingsvolgorde af te dwingen is het gebruik van een compleet stel (eventueel geneste) haakjes:
- Vb2 y=a+b+c+d+e+f-g-h -> (((a+(b+c))+d)+((e+(f-g))-h))
- Elk stel haakjes geeft een niveau in de expressieboom aan en laat alleen deze expressieboom toe
- Op elk gelijk niveau haakjes is maar 1 operatie mogelijk



Afdwingen bepaalde expresieboom zonder gebruik van haakjes

- Prefix, infix en postfix notatie van ((a-(b*c))+d):
  - Prefix: + - a * b c d Operatie Op1 Op2 trigger
  - Infix: a – b * c + d Op1 Operatie Op2 trigger
  - Postfix: a b c * - d + Op1 Op2 Operatie trigger



- preOrder, inOrder en postOrder doorloop EB:
  - PreOrder: + - a * b c d
  - InOrder : a – b * c + d
  - PostOrder: a b c * - d + Reverse Polish Notation RPN

Stack en afhandelingsvolgorde RPN

- De stack kan bij RPN heel handig gebruikt worden om:
- operanden op de stack te pushen en om:
- bij langskomen van een operatie:
  - de 1 of 2 benodigde operanden van de stack te poppen,
  - de operatie uit te voeren en
  - het resultaat weer op de stack te pushen

  - Dit werd veel gebruikt in vroege types digitale rekenmachines

Infix notatie en inOrder doorloop

- Gebruikers zijn gewend aan infix notatie die hoort bij een inOrder doorloop van de EB:
- Bij de doorloop kun je steeds een operatie uitvoeren als de benodigde 1 of 2 operand waarden er kinderen van zijn
- De kind(eren) die nog geen operand zijn leiden tot een recursiestap, die tot herhaalde recursie kan leiden
- Recursie voert als het ware automatisch backtracking uit na het op blad niveau oplossen van een operatie